Apache-mod_ssl/SSL

The Secure Sockets Layer protocol is a protocol layer which may be placed between a reliable connection-oriented network layer protocol (e.g. TCP/IP) and the application protocol layer (e.g. HTTP). SSL provides for secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity, and encryption for privacy.

The protocol is designed to support a range of choices for specific algorithms used for cryptography, digests, and signatures. This allows algorithm selection for specific servers to be made based on legal, export or other concerns, and also enables the protocol to take advantage of new algorithms. Choices are negotiated between client and server at the start of establishing a protocol session.

There are a number of versions of the SSL protocol, as shown As noted below, one of the benefits in SSL 3.0 is that it adds support of certificate chain loading. This feature allows a server to pass a server certificate along with issuer certificates to the browser. Chain loading also permits the browser to validate the server certificate, even if Certificate Authority certificates are not installed for the intermediate issuers, since they are included in the certificate chain. SSL 3.0 is the basis for the Transport Layer Security [TLS] protocol standard.

SSL v2.0 Vendor Standard (from Netscape Corp.) [SSL2] First SSL protocol for which implementations exists - NS Navigator 1.x/2.x - MS IE 3.x - Lynx/2.8+OpenSSL

SSL v3.0 Expired Internet Draft (from Netscape Corp.) [SSL3] Revisions to prevent specific security attacks, add non-RSA ciphers, and support for certificate chains - NS Navigator 2.x/3.x/4.x - MS IE 3.x/4.x – Lynx/2.8+OpenSSL

TLS v1.0 Proposed Internet Standard (from IETF) [TLS1] Revision of SSL 3.0 to update the MAC layer to HMAC, add block padding for block ciphers, message order standardization and more alert messages.

SSL v2.0 Vendor Standard (from Netscape Corp.) [

Session Establishment

The SSL session is established by following a handshake sequence between client and server.Once an SSL session has been established it may be reused, thus avoiding the performance penalty of repeating the many steps needed to start a session. For this the server assigns each SSL session a unique session identifier which is cached in the server and which the client can use on forthcoming connections to reduce the handshake (until the session identifer expires in the cache of the server). The elements of the handshake sequence, as used by the client and server, are listed below:

1. Negotiate the Cipher Suite to be used during data transfer
2. Establish and share a session key between client and server
3. Optionally authenticate the server to the client
4. Optionally authenticate the client to the server

The first step, Cipher Suite Negotiation, allows the client and server to choose a Cipher Suite supportable by both of them. The SSL3.0 protocol specification defines 31 Cipher Suites. A Cipher Suite is defined by the following components:

*** Key Exchange Method**
*** Cipher for Data Transfer**
*** Message Digest for creating the Message Authentication Code (MAC)**

These three elements are described in the sections that follow.

Key exchange method

The key exchange method defines how the shared secret symmetric cryptography key used for application data transfer will be agreed upon by client and server. SSL 2.0 uses RSA key exchange only, while SSL 3.0 supports a choice of key exchange algorithms including the RSA key exchange when certificates are used, and Diffie-Hellman key exchange for exchanging keys without certificates and without prior communication between client and server.

One variable in the choice of key exchange methods is digital signatures -- whether or not to use them, and if so, what kind of signatures to use. Signing with a private key provides assurance against a man-in-the-middle-attack during the information exchange used in generating the shared key.

Cipher for data transfer

SSL uses symmetric cryptography for encrypting messages in a session. There are nine choices, including the choice to perform no encryption:

* No encryption
* Stream Ciphers
  o RC4 with 40-bit keys
  o RC4 with 128-bit keys
* CBC Block Ciphers
  o RC2 with 40 bit key
  o DES with 40 bit key
  o DES with 56 bit key
  o Triple-DES with 168 bit key
  o Idea (128 bit key)
  o Fortezza (96 bit key)

Here "CBC" refers to Cipher Block Chaining, which means that a portion of the previously encrypted cipher text is used in the encryption of the current block. "DES" refers to the Data Encryption Standard, which has a number of variants (including DES40 and 3DES_EDE). "Idea" is one of the best and cryptographically strongest available algorithms, and "RC2" is a proprietary algorithm from RSA DSI

Digest Function

The choice of digest function determines how a digest is created from a record unit. SSL supports the following:

*** No digest (Null choice)**
*** MD5, a 128-bit hash**
*** Secure Hash Algorithm (SHA-1), a 160-bit hash**

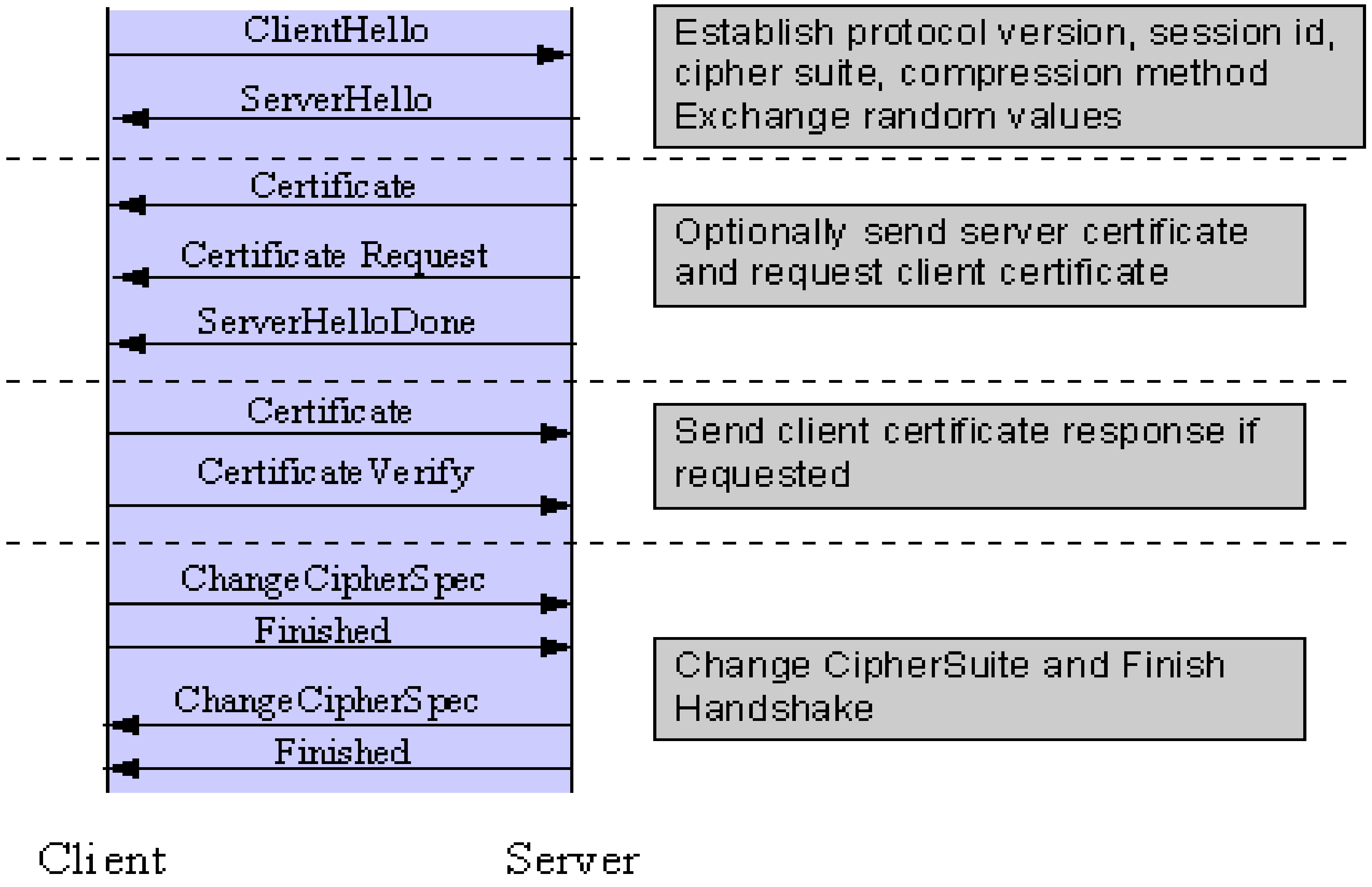
The message digest is used to create a Message Authentication Code (MAC) which is encrypted with the message to provide integrity and to prevent against replay attacks.

Handshake Sequence Protocol

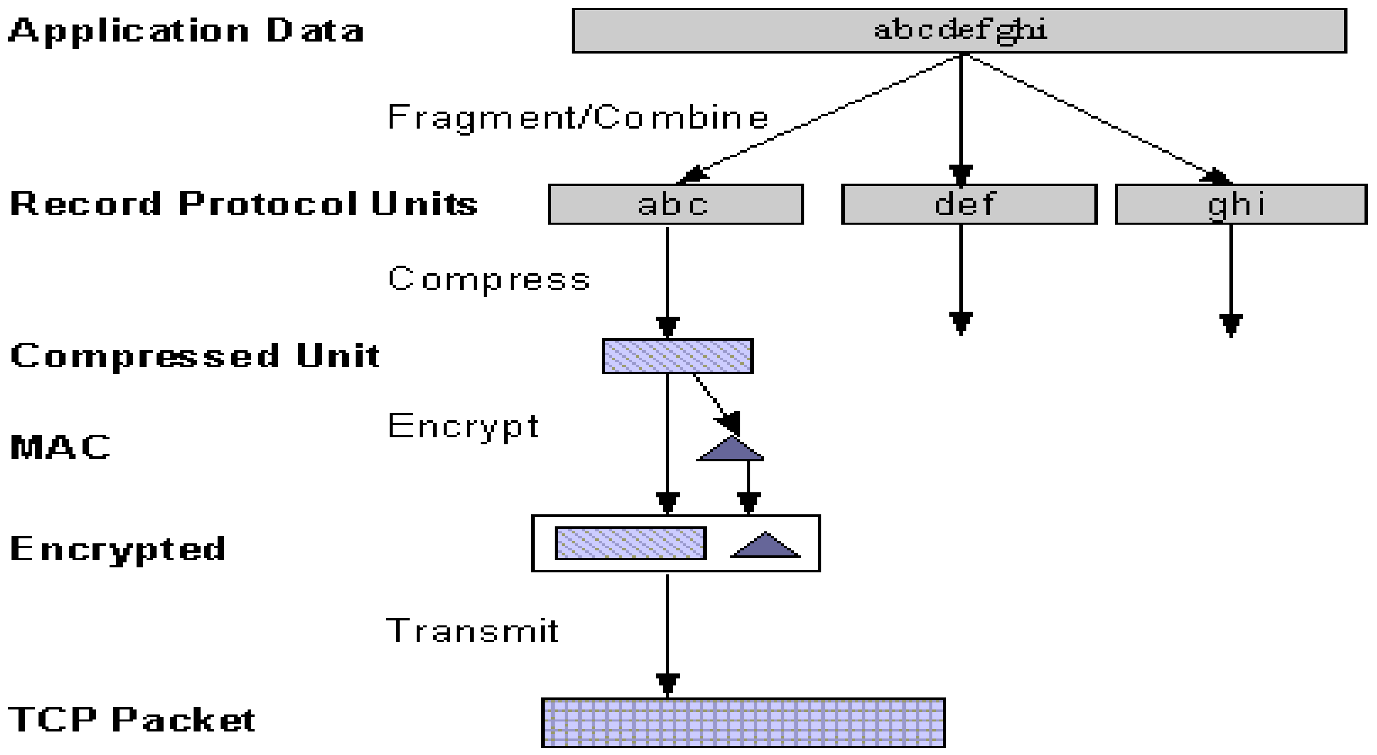
The handshake sequence uses three protocols:

*** The SSL Handshake Protocol for performing the client and server SSL session establishment.**
*** The SSL Change Cipher Spec Protocol for actually establishing agreement on the Cipher Suite for the session.**
*** The SSL Alert Protocol for conveying SSL error messages between client and server.**

These protocols, as well as application protocol data, are encapsulated in the SSL Record Protocol. An encapsulated protocol is transferred as data by the lower layer protocol, which does not examine the data. The encapsulated protocol has no knowledge of the underlying protocol.
The encapsulation of SSL control protocols by the record protocol means that if an active session is renegotiated the control protocols will be transmitted securely. If there were no session before, then the Null cipher suite is used, which means there is no encryption and messages have no integrity digests until the session has been established.


ClientHello
ServerHello
Establish protocol version, session id, cipher suite, compression method Exchange random values
Certificate
Certificate Request
ServerHelloDone
Optionally send server certificate and request client certificate
Certificate
CertificateVerify
Send client certificate response if requested
ChangeCipherSpec
Finished
ChangeCipherSpec
Finished
Change CipherSuite and Finish Handshake
Client
Server

The SSL Record Protocol is used to transfer application and SSL Control data between the client and server, possibly fragmenting this data into smaller units, or combining multiple higher level protocol data messages into single units. It may compress, attach digest signatures, and encrypt these units before transmitting them using the underlying reliable transport protocol.



One common use of SSL is to secure Web HTTP communication between a browser and a webserver. We also commonly use it to secure other applications like pop3 and imap.

Installing apache with mod_ssl

Ftp to noc.e0.ws.afnog.org and pick the package from
/pub/FreeBSD/i386/releases/5.3-RELEASE/packages/www/apache13-modssl.tbz
In our case we already have a previously installed version of apache that does not have ssl support.
We need to delete this package first before we can install the new version.
First we need to see what the installed version is using:

```
pkg_info |grep apache
```

This will return

```
apache-1.3.33 ( and some description here)
```

We then delete apache using

```
pkg_delete apache-1.3.33
```

We can now install our new version of apache that we have downloaded using

```
pkg_add apache13-modssl.tbz
```

You can also install apache using the ports system by doing the following steps.

```
cd /usr/ports/www/apache-modssl
make install
```

This will download, compile and install apache with ssl support. We shall cover this in detail when learning about the ports system.

Checking that Apache is working

After we have installed our apache webserver we now need to start it up and check that it is running:

```
/usr/local/etc/rc.d/apache.sh start
```

This starts apache

We then check that it is running by using:

```
ps -auxw |grep httpd
```

You should see output similar to

```
root      432  0.0  0.3  5296   744  ??  Ss   11:15AM   0:01.40 /usr/local/sbin/httpd -DSSL
www       515  0.0  0.7  5372  1716  ??  I    11:15AM   0:00.01 /usr/local/sbin/httpd -DSSL
www       516  0.0  0.5  5384  1352  ??  I    11:15AM   0:00.00 /usr/local/sbin/httpd -DSSL
www       517  0.0  0.5  5324  1252  ??  I    11:15AM   0:00.00 /usr/local/sbin/httpd -DSSL
www       518  0.0  0.7  5356  1708  ??  I    11:15AM   0:00.00 /usr/local/sbin/httpd -DSSL
www       519  0.0  0.7  5356  1708  ??  I    11:15AM   0:00.00 /usr/local/sbin/httpd -DSSL
www      1437  0.0  0.7  5372  1708  ??  I     2:11PM   0:00.00 /usr/local/sbin/httpd -DSSL
```

Which shows that apache is running with ssl support.

We can also check that there is actually a listening process on both port 80 for normal http and 443 for ssl by running

```
netstat -an |grep LIST
tcp4    0    0  *.110          *.*          LISTEN
tcp4    0    0  *.25           *.*          LISTEN
tcp4    0    0  *.21           *.*          LISTEN
tcp4    0    0  *.5999         *.*          LISTEN
tcp4    0    0  *.80           *.*          LISTEN
tcp4    0    0  *.443          *.*          LISTEN
tcp4    0    0  *.22           *.*          LISTEN
```

Testing our apache webserver

You can now check your apache webserver using both lynx-ssl and the openssl s_client

```
[inst@noc inst]$ openssl
OpenSSL> s_client -host localhost -port 443
```

This will output what happens when you make an ssl connection and show us details of the certificates installed. We can also do this easier using

```
lynx https://localhost
```

Certificate management

We shall now setup a certificate for our server in order to begin serving https requests.
For this purpose we shall use the CA.sh or CA.pl scripts that come with openssl.
First we need to create our own Certificate Authority
This we do by running

1. cd /usr/src/crypto/openssl/apps
2. ./CA.sh -newca

You will be prompted for a pass phrase which will be used to protect your new certificate for the authority.
Once this is done,We will then generate a signing request which we will then sign with our new Certificate Authority.

./CA.sh -newreq to generate a new request for signing and
CA.sh -sign in order to sign the request.
The next step is then to setup apache to use our new certificate. This is done by editing httpd.conf and changing
SSLCertificateFile
SSLCertificateKeyFile
to point to our new certificate and its key.
In normal production environments, we would normally only generate a signing requestthat would then be signed by a recognised certificate authority.

Removing the pass-phrase from the Certificate

The RSA private key inside your server.key file is stored in encrypted format for security reasons. The pass-phrase is needed to be able to read and parse this file thus everytime you want to start the apache server you would need to enter your passphrase. This is not useful if you want to remotely reboot your server or if you're likely to get power issues so once you've secured your server , you can then remove the passphrase so you can automatically start apache at boot.

1. cp serverkey.pem serverkey.pem.orig
2. openssl rsa -in serverkey.pem.orig -out serverkey.pem

You can now remove the directive that points to the certificate key as you no longer need it now. It is also advisable to change the permissions on the key to make it readable only by root since it is no longer encrypted. You should now be able to restart apache without having to enter a pass phrase.

Notes

Name based virtual hosts with apache will not work with SSL. You need to use ip based virtual host because name based hosts are implemented at the application layer while TLS is at the transport layer. Documentation can be found at

http://httpd.apache.org/docs

http://www.modssl.org